

Manipulacija podacima u relacionoj bazi

Četiri osnovne operacije za manipulaciju podacima u relacionim bazama podataka su umetanje, prikazivanje, ažuriranje i brisanje. Ove operacije se u računarskom programiranju nazivaju **CRUD operacije**. Akronim *CRUD* (*Create, Read, Update, Delete*) odnosi se na glavne funkcije koje su implementirane u aplikacijama za relacione baze podataka. Ove operacije pripadaju **DML (data manipulation language)** jeziku za komunikaciju sa bazama podataka, koji predstavlja deo integrisanog SQL jezika upita. U nastavku ćemo prikazati naredbe koje se u SQL-u koriste za izvršavanje ovih operacija (tabela 6.1).

Naredbe za CRUD operacije u SQL jeziku upita		
Operacija	SQL naredba	Funkcija
Create	INSERT	umetanje podataka
Read (Retrieve)	SELECT	prikazivanje podataka
Update (Modify)	UPDATE	ažuriranje podataka
Delete (Destroy)	DELETE	brisanje podataka

Tabela 6.1. Naredbe za CRUD operacije u SQL jeziku upita

Kako bismo prikazali upotrebu ovih operacija nad bazom, za primer ćemo uzeti bazu podataka koju smo kreirali u prethodnoj lekciji korišćenjem `mysql.connector` modula. U nastavku je priložen kod koji smo koristili za kreiranje naše baze.

Primer kreiranja baze podataka korišćenjem `execute()` metode `sqlite3` modula:

```
import mysql.connector

mydb = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password123",
    database="mydb"
)

cursor = mydb.cursor()

cursor.execute("""CREATE TABLE IF NOT EXISTS Toy_Store_Products(
    toy_id INT PRIMARY KEY,
    product_name TEXT,
    price FLOAT,
    quantity INT);
""")

mydb.commit()
```

Umetanje podataka u bazu (INSERT)

Za umetanje podataka u bazu takođe koristimo objekat `cursor` i njegovu metodu `execute()`. Upit, odnosno naredba koju prosleđujemo kao parametar za upisivanje podataka, mora biti lista n-torki, string ili rečnik, u zavisnosti od toga da li je dodavanje podataka parametrizovano ili ne. U nastavku ćemo prikazati neparemetrizovano dodavanje proizvoda u tabelu lager liste u prodavnici igračaka korišćenjem SQL upita.

Primer neparametrizovanog umetanja novog reda u bazu podataka korišćenjem INSERT naredbe execute() metode sqlite3 modula:

```
toy = ('100', 'Teddy Bear', 25.0, 40)
cursor.execute("INSERT INTO Toy_Store_Products VALUES (?, ?, ?, ?)",
toy)

mydb.commit()
```

Izmeniti kod tako da se u bazi `example.db` napravite tabelu `Book` pomoću `sqlite3` modula koja bi trebalo da sadrži attribute: `book_id` (ceo broj, predstavlja primarni ključ za ovu tabelu), `writer` (string), `genre` (string), `year` (ceo broj) Unesite 2 knjige u bazu podataka.

Radno okruženje

Upit ćemo opet razložiti na više celina radi bolje preglednosti:

- `INSERT INTO` – obavezna naredba kojom se označava upisivanje u tabelu;
- `Toy_Store_Products` – naziv tabele u koju vršimo upisivanje;
- `VALUES(?, ?, ?, ?)` – rezervisana porcija za prosleđivanje n-torke podataka; svaki znak `?` predstavlja po jedno rezervisano mesto (placeholder) za svaki atribut iz n-torke koja će biti prosleđena; ukoliko se broj atributa ne poklapa sa brojem placeholdera, doći će do greške.

Parametrizovano dodavanje podataka koristi se kada radimo sa podacima skladištenim u rečniku. U nastavku ćemo prikazati kako bi izgledalo parametrizovano dodavanje podataka na identičnom primeru upisivanja proizvoda na lager prodavnice igračaka.

Primer parametrizovanog dodavanja novog reda u bazu podataka korišćenjem INSERT naredbe i execute() metode sqlite3 modula:

```
toy = {"toy_id": '100', 'product_name': 'Teddy Bear',
"price": '25.0', "quantity": '40'}

cursor.execute("INSERT INTO Toy_Store_Products VALUES (:toy_id,
:product_name, :price, :quantity)", toy)

conn.commit()
```

Za razliku od prethodne metode dodavanja, kod parametrizacije su atributi skladišteni u rečniku u parovima ključ-vrednost. Za prosleđivanje vrednosti atributa neophodno je naznačiti njegov ključ sa dve tačke (:) kao prefiksom.

Kod SQL upita u sqlite3 modulu podržana je metoda koja omogućava izvršavanje višestrukih naredbi. Korišćenjem `executemany()` prikazaćemo kako na našu lager listu proizvoda u prodavnici igračaka dodajemo dva nova proizvoda. Ono što je bitno napomenuti je da u ovoj metodi podaci koji se prosleđuju moraju biti lista n-torki ili lista rečnika.

Primer neparametrizovanog dodavanja dva reda u bazu podataka korišćenjem INSERT naredbe i `executemany()` metode `sqlite3` modula:

```
toy = [('101', 'Bicycle', 100.0, 15), ('102', 'Doll', 20.0, 35)]
cursor.executemany("INSERT INTO Toy_Store_Products VALUES (?, ?, ?, ?)",
    toy)

conn.commit()
```

Vraćanje podataka iz baze (SELECT)

U metodi `execute()` modula `sqlite3` možemo proslediti i upit koji će prema zadatom uslovu čitati podatke iz naše baze. Međutim, kako bi naša metoda uzela sve redove koje uslov za prikaz obuhvata, neophodno je pozivanje metode `fetchall()` objekta kursora. Pogledajmo u nastavku šta se sve nalazi na našoj lager listi prodavnice igračaka.

Primer pregleda svih redova tabele korišćenjem SELECT naredbe i `executemany()` i `fetchall()` metode `sqlite3` modula:

```
cursor.execute("SELECT * FROM Toy_Store_Products;")
product_list = cursor.fetchall()

print(product_list)
```

Podela upita na celine izgledala bi ovako:

- `SELECT` – obavezna naredba za prikaz podataka;
- – podatak ili set podataka koje želimo da prikažemo. Znak `*` je oznaka koja zamenjuje sve redove tabele;
- `FROM` – naziv tabele iz koje želimo da prikažemo podatke.

```

In [9]: import sqlite3

conn = sqlite3.connect('example.db')
cursor = conn.cursor()

cursor.execute("""CREATE TABLE IF NOT EXISTS Toy_Store_Products(
toy_id INT PRIMARY KEY,
product_name TEXT,
price FLOAT,
quantity INT);
""")

toy = ('100', 'Teddy Bear', 25.0, 40)
cursor.execute("INSERT INTO Toy_Store_Products VALUES (?, ?, ?, ?)", toy)

toy = [('101', 'Bicycle', 100.0, 15), ('102', 'Doll', 20.0, 35)]
cursor.executemany("INSERT INTO Toy_Store_Products VALUES (?, ?, ?, ?)", toy)

cursor.execute("SELECT * FROM Toy_Store_Products;")
product_list = cursor.fetchall()

conn.commit()

print(product_list)

```

[(100, 'Teddy Bear', 25.0, 40), (101, 'Bicycle', 100.0, 15), (102, 'Doll', 20.0, 35)]

Slika 6.1. Prikaz kreirane tabele sa podacima example.db baze podataka u Jupyter Notebook Okruženju

Primer pravljenje baze puter sqlite3 modula, unos podataka i ispit podataka iz baze.

Radno okruženje

```

import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()

cursor.execute("""CREATE TABLE IF NOT EXISTS Toy_Store_Products(
    toy_id INT PRIMARY KEY,
    product_name TEXT,
    price FLOAT,
    quantity INT);
""")

conn.commit()

toy = ('100', 'Teddy Bear', 25.0, 40)
cursor.execute("INSERT INTO Toy_Store_Products VALUES (?, ?, ?, ?)",
toy)

conn.commit()

cursor.execute("SELECT * FROM Toy_Store_Products;")
product_list = cursor.fetchall()

print(product_list)

```

Izmeniti kod tako da se u bazi example.db napravite tabelu Fruit pomoću sqlite3 modula koja bi trebalo da sadrži attribute: barcode(ceo broj, predstavlja primarni ključ za ovu tabelu), name(string), sort(string), price(realan broj). Unesite podatke za 3 voca pomoću executemany metode i ispišite podatke iz baze na standardni izlaz.

Ažuriranje baze podataka (UPDATE)

Ako posmatramo našu prodavnicu igračaka u realnom svetu, zaključićemo da njena lager lista mora da bude podložna promenama u smislu ažuriranja podataka. SQL jezici podržavaju i ovu funkcionalnost. Recimo da želimo da prodamo što više bicikala pre nego što stigne zima. U skladu sa ovim snizićemo njihovu cenu na 80 dolara. Evo kako to izgleda u sqlite3 modulu.

Primer ažuriranja baze podataka korišćenjem UPDATE naredbe i execute(metode sqlite3 modula:

```
cursor.execute("""UPDATE Toy_Store_Products SET price = 80.0
WHERE product_name = 'Bicycle'""")

conn.commit()
```

Pogledajmo kako izgleda naš upit po celinama:

- UPDATE – obavezna naredba koja označava ažuriranje tabele;
- SET – ključna reč za postavljanje nove vrednosti;
- WHERE – ključna reč za postavljanje uslova u kojima će se nova vrednost postaviti.

Izmenite prethodni zadatak tako da se u tabeli Book podatak year knjige sa definisanim podatkom id_knjige 5 menja na 2020.

Pitanje

Slovo C u akronimu *CRUD* predstavlja SQL komandu:

- CREATE
- INSERT
- UPDATE
- DELETE

Objašnjenje:

Slovo C u akronimu *CRUD* predstavlja *INSERT* komandu SQL upitnog jezika, odnosno komandu umetanja.

Brisanje podataka iz baze (DELETE)

Još jedna funkcija koju SQL jezici podržavaju je funkcija brisanja. Ako bismo želeli da obrišemo lutke iz asortimana proizvoda, to bismo uradili na sledeći način:

Primer brisanja podatka iz baze korišćenjem DELETE naredbe i execute() metode sqlite3 modula:

```
cursor.execute("""DELETE FROM Toy_Store_Products WHERE product_name =
'Doll' """)

conn.commit()
```

Upit ćemo, radi bolje preglednosti, i ovog puta razložiti:

- DELETE FROM – obavezna naredba koja označava brisanje podatka;
- WHERE – ključna reč za postavljanje uslova u kojima će se vrednost obrisati.

Pogledajmo kako sada izgleda naša baza, posle ovih promena. Pregled vršimo pomoću komande SELECT *, kao što je to ranije u lekciji objašnjeno.

```
In [2]: import sqlite3

conn = sqlite3.connect('example.db')

cursor = conn.cursor()

cursor.execute("""CREATE TABLE IF NOT EXISTS Toy_Store_Products(
toy_id INT PRIMARY KEY,
product_name TEXT,
price FLOAT,
quantity INT);
""")

toy = ('100', 'Teddy Bear', 25.0, 40)
cursor.execute("INSERT INTO Toy_Store_Products VALUES (?, ?, ?, ?)", toy)

toy = [('101', 'Bicycle', 100.0, 15), ('102', 'Doll', 20.0, 35)]
cursor.executemany("INSERT INTO Toy_Store_Products VALUES (?, ?, ?, ?)", toy)

cursor.execute("""UPDATE Toy_Store_Products SET price = 80.0
WHERE product_name = 'Bicycle'""")

cursor.execute("""DELETE FROM Toy_Store_Products
WHERE product_name = 'Doll'""")

cursor.execute("SELECT * FROM Toy_Store_Products;")
product_list = cursor.fetchall()

conn.commit()

print(product_list)

[(100, 'Teddy Bear', 25.0, 40), (101, 'Bicycle', 80.0, 15)]
```

Slika 6.2. Prikaz ažuriranja tabele sa podacima example.db baze podataka u Jupyter Notebook okruženju

Izmeniti kod tako da se u bazi example.db napravite tabelu Fruit pomoću sqlite3 modula koja bi trebalo da sadrži atribute: barcode(ceo broj, predstavlja primarni ključ za ovu tabelu), name(string), sort(string), price (realan broj). Unesite podatke za 3 voća pomoću executemany metode i ispišite podatke iz baze na standardni izlaz. Povećati cenu za jedno voće. Izbrisati voće koje je prvo dodato.

CRUD operacije u mysql modulu

CRUD operacije u mysql modulu primenjuju se identično kao i u ostalim modulima relacionih baza podataka. Prikazaćemo samo neke od ovih operacija na poznatom primeru prodavnice igračaka. Podsetimo se kako smo izvršili konekciju na bazu podataka i kreirali tabelu.

Primer kreiranja tabelle korišćenjem `execute()` metode mysql modula:

```
import mysql.connector

conn = mysql.connector.connect(user='root', database='example')

cursor = conn.cursor()
cursor.execute("""CREATE TABLE IF NOT EXISTS Toy_Store_Products(
    toy_id INT PRIMARY KEY,
    product_name TEXT,
    price FLOAT,
    quantity INT);
""")

conn.commit()
```

Tabela koju smo kreirali je trenutno prazna. Na našu lager listu ćemo odjednom dodati sve proizvode koji postoje i u našem primeru u `sqlite3` modulu. Identičnim postupkom prosleđujemo upit `execute()` metodi korišćenjem ključnih reči za operaciju upisivanja `INSERT INTO`. Međutim, postoji mala razlika u ovom modulu, koja se tiče rezervisanja porcije za prosleđivanje *n*-torke podataka. Ovaj modul ne podržava `?` kao karakter koji predstavlja placeholder, već se umesto toga koriste `%s` kao placeholderi za rezervisane podatke koji se u MySQL upitu uvek prosleđuju u vidu stringova. U nastavku je prikazan deo koda koji će izvršiti naredbu umetanja.

Primer neparametrizovanog dodavanja više redova u bazu podataka korišćenjem `INSERT` naredbe i `executemany()` metode mysql modula:

```
toy = [('100', 'Teddy Bear', 25.0, 40),
       ('101', 'Bicycle', 100.0, 15),
       ('102', 'Doll', 20.0, 35)]

cursor.executemany("INSERT INTO Toy_Store_Products VALUES (?, ?, ?, ?)",
                  toy)

conn.commit()
```

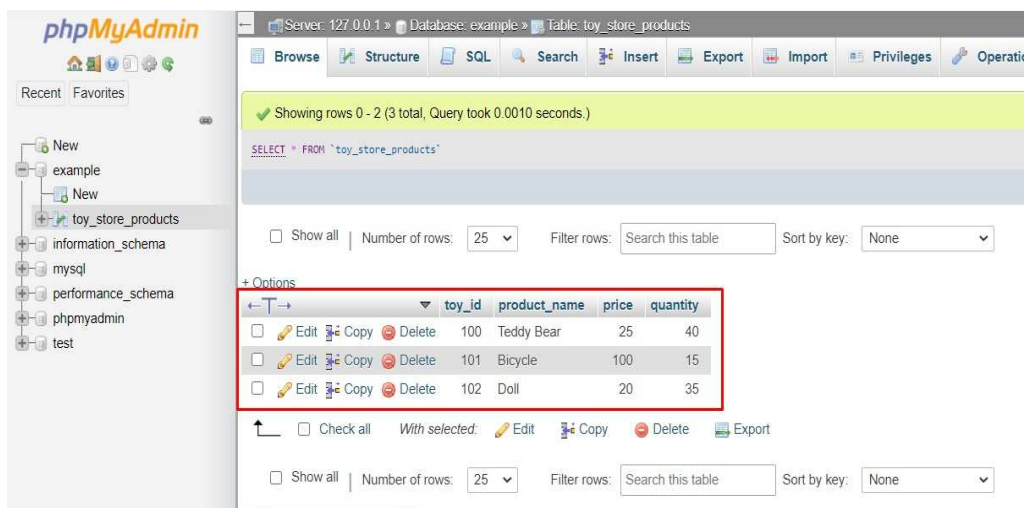
Napomena

*Postoji i drugi način umetanja podataka u bazu bez specificiranja rezervisanih tipova podataka u porciji za prosleđivanje *n*-torke. Ovo je moguće postići podešavanjem parametra kursora na vrednost `True`, što bi izgledalo ovako:*

```
cursor = conn.cursor(prepared=True)
```

Ovaj parametar omogućava da se isti upit efikasno izvršava više puta tako što ima pripremljen obrazac upita koji ponavlja svakim upitom.

Izvršavanjem koda priloženog u posljednjem primeru, u tabelu smo uneli tri stavke. Za pregled ovih stavki moguće je koristiti CRUD operaciju `SELECT` za prikazivanje podataka ili jednostavno fizički pristupiti MySQL klijentu. U nastavku prikazujemo kako pregled ovih podataka izgleda korišćenjem oba pristupa.



Showing rows 0 - 2 (3 total, Query took 0.0010 seconds.)

`SELECT * FROM 'toy_store_products'`

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

+ Options

	toy_id	product_name	price	quantity
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	100	Teddy Bear	25	40
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	101	Bicycle	100	15
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	102	Doll	20	35

☐ Check all | With selected: ☐ Edit ☐ Copy ☐ Delete ☐ Export

☐ Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

Slika 6.3. Prikaz ažurirane tabele `toy_store_products` tabele

Primer pregleda svih redova tabele korišćenjem `SELECT` naredbe i `executemany()` i `fetchall()` metoda `mysql` modula:

```
cursor.execute("SELECT * FROM Toy_Store_Products;")
product_list = cursor.fetchall()

print(product_list)
```



```

In [3]: import mysql.connector

conn = mysql.connector.connect(user='root', database='example')

cursor = conn.cursor(prepared=True)

cursor.execute("""CREATE TABLE IF NOT EXISTS Toy_Store_Products(
toy_id INT PRIMARY KEY,
product_name TEXT,
price FLOAT,
quantity INT);
""")

cursor.execute("SELECT * FROM Toy_Store_Products;")
product_list = cursor.fetchall()

toy = [('100', 'Teddy Bear', 25.0, 40),
('101', 'Bicycle', 100.0, 15),
('102', 'Doll', 20.0, 35)]

cursor.executemany("INSERT INTO Toy_Store_Products VALUES (?, ?, ?, ?)", toy)

conn.commit()

print(product_list)

[(100, 'Teddy Bear', 25.0, 40), (101, 'Bicycle', 100.0, 15), (102, 'Doll', 20.0, 35)]

```

Slika 6.4. Rezultat prikazivanja svih podataka tabele baze example.db u Jupyter Notebook okruženju

+ Options

	toy_id	product_name	price	quantity
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	100	Teddy Bear	25	40
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	101	Bicycle	100	15
<input type="checkbox"/> Edit <input type="checkbox"/> Copy <input type="checkbox"/> Delete	102	Doll	20	35

☐ Check all With selected: ☐ Edit ☐ Copy ☐ Delete ☐ Export

☐ Show all Number of rows: 25 Filter rows: Search this table

Slika 6.5. Rezultat prikazivanja svih podataka tabele baze example.db u phpMyAdmin okruženju

Zadaci za vežbu

Probajte samostalno da uradite ove zadatke:

Radno okruženje

Zadatak 1

U bazi example.db napravite tabelu Company pomoću sqlite3 modula koja bi trebalo da sadrži atribute: company_id(ceo broj, predstavlja primarni ključ za ovu tabelu), owner (string), director(string), city(string), adress(string), phone_number(string), email(string). Unesite podatke za 3 kompanije, izmenite podatak email adrese za kompaniju po želji a poslednju dodatu kompaniju izbrišite iz baze. Ispišite podatke iz baze na standarni izlaz.

Rešenje:

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()

cursor.execute("""CREATE TABLE IF NOT EXISTS Company(
    company_id INT PRIMARY KEY,
    owner TEXT,
    director TEXT,
    city TEXT,
    adress TEXT,
    phone_number TEXT,
    email TEXT);
""")

conn.commit()

companys= [(101, 'NN', 'MM','BG','Pere 12','+51545','mdksalm@mdkslds'),
(102, 'NM', 'MN','BG','Pere 13','+5154532','mdksalm@mdksldsads'),(103,
'KK', 'SM','BG','Pere 19','+51545153','a@a')]
cursor.executemany("INSERT INTO Company VALUES (?, ?, ?, ?, ?, ?, ?)",companys)
conn.commit()

cursor.execute("""UPDATE Company SET email = 'email.email' WHERE
company_id = 101""")
conn.commit()

cursor.execute("""DELETE FROM Company WHERE company_id = 103""")
conn.commit()

cursor.execute("SELECT * FROM Company;")
company_list= cursor.fetchall()

print(company_list)
```

Objašnjenje:

Prvi korak jeste uključivanje potrebnog `mysql.connector` modula, kreiranje konekcije i kursor objekta. Nakon toga, potrebno je da izvršimo `select` upit gde prosleđujemo ime tabele. Na samom kraju, petljom prolazimo kroz sve rezultate liste pojedinačno i ispisujemo ih u pogodnom formatu, koristeći `format` funkciju.

Zadatak 2

Potrebno je omogućiti korisniku da unese odvojeno JMBG, ime, prezime i adresu koristeći komandnu liniju. Unete podatke je potrebno uskladištiti u SQLite bazu podataka. Nakon skladištenja, potrebno je korisniku ispisati sve podatke iz tabele.

Rešenje:

```
import sqlite3
conn = sqlite3.connect('example.db')
cursor = conn.cursor()

cursor.execute("""CREATE TABLE IF NOT EXISTS user(
    user_id INT PRIMARY KEY,
    name TEXT,
    last_name TEXT,
    address TEXT);
""")
conn.commit()

user_id = input("Enter your ID:")
name = input("Enter your name:")
last_name = input("Enter your last name:")
address = input("Enter your address:")

user = (user_id, name, last_name, address)
cursor.execute("INSERT INTO user VALUES (?, ?, ?, ?)", user)

conn.commit()

cursor.execute("SELECT * FROM user;")
product_list = cursor.fetchall()
for p in product_list:
    print("ID: {} | Name: {} | Last name: {} | Address: {}".format(p[0],
p[1], p[2], p[3]))
```

Objašnjenje:

Prvi korak jeste uključivanje potrebnog `sqlite3` modula i kreiranje konekcije i kursor objekta. Nakon toga, potrebno je da kreiramo tabelu koristeći `execute` metodu kursor objekta. Zatim, koristeći `input()` funkciju, možemo od korisnika zatražiti unos podataka i te podatke sačuvati. Kada su svi podaci popunjeni, kreiramo `user` n-torku koju ćemo iskoristiti prilikom kreiranja `insert` upita. Na samom kraju, `SELECT` upitom dobavljamo sve podatke iz baze i prikazujemo ih korisniku.

Rezime

- Za manipulaciju podacima u bazama podataka koristi se DML (data manipulation language).
- Četiri osnovne operacije za manipulaciju su: umetanje, prikazivanje, ažuriranje i brisanje, koje jednim imenom nazivamo *CRUD operacije*.
- Operacija umetanja koristi ključne reči `INSERT INTO`, koje prati naziv tabele u koju želimo da umetnemo podatke.
- U drugom delu upita za neparametrizovano umetanje podataka definiše se rezervisana porcija za prosleđivanje n-torke podataka koja se u `sqlite3` modulu označava `VALUES(?, ?, ?, ?)`, a u `mysql` modulu `VALUES(%s, %s, %s, %s)`. Drugi način u `mysql` modulu je podešavanje parametra kursora `prepared` na vrednost `True`.

- Parametrizovano umetanje podataka koristi se za umetanje podataka koji su definisani u rečniku. Rezervisana porcija za prosleđivanje n-torke podataka u tom slučaju za placeholdera postavlja ključeve podataka `VALUES (:key, :key, :key, :key)`.
- Operacija prikazivanja koristi ključnu reč `SELECT` koju prati naziv atributa, a zatim se pomoću ključne reči `FROM` naznačava tabela iz koje će podaci biti pročitani.
- Operacija ažuriranja koristi ključnu reč `UPDATE`, koju prati naziv tabele, a zatim se pomoću ključne reči `SET` određuje nova vrednost atributa i, konačno, pomoću `WHERE` uslova, određuje se pod kojim okolnostima će se određene promene aplicirati.
- Operacija brisanja koristi ključne reči `DELETE FROM`, koje prati naziv tabele, a zatim pomoću `WHERE` uslova definišemo pod kojim okolnostima će se brisanje izvršiti.

